**Задача 1**. Фигуры

**Часть 1.** Создать иерархию классов для геометрических фигур.

Для них нужно создать интерфейс Shape, в котором определены методы:

double getWidth()  
double getHeight()  
double getArea()  
double getPerimeter()

Все фигуры должны реализовывать этот интерфейс и правильным образом определять данные методы

1. Square – квадрат

Должен иметь конструктор, принимающий длину стороны

1. Triangle – треугольник

Должен иметь конструктор, принимающий x1, y1, x2, y2, x3, y3 – шесть координат.  
В качестве ширины возвращать max(x1, x2, x3) – min(x1, x2, x3)  
В качестве высоты возвращать max(y1, y2, y3) – min(y1, y2, y3)

1. Rectangle – прямоугольник

Должен иметь конструктор, принимающий длины двух сторон

1. Circle – окружность

Должна иметь конструктор, принимающий радиус. В качестве ширины и высоты должен выдаваться диаметр

**Часть 2.** В main в коде объявить список фигур, чтобы в нём было около 5-10 разных фигур. Задача – написать функцию, которая находит фигуру с максимальной площадью. Вызвать её для этого списка и распечатать информацию о фигуре в консоль.

Аналогично найдите фигуру со вторым по величине периметром.

Поиск фигур реализовать через стандартный метод Arrays.sort (в C# Array.Sort) с компаратором. Что такое компаратор почитайте сами, но если будут вопросы, задавайте.

**Часть 3.** Переопределите в фигурах методы toString, hashCode, equals

Чему научитесь:

* Интерфейсы, реализация интерфейса
* Виртуальные функции – как они работают и применяются
* Реализация toString, hashCode, equals
* Применение стандартной сортировки, компараторы, реализация интерфейсов

**Задача 2.** Вектор

Реализовать класс Vector для векторов вещественных чисел размерности n. Компоненты вектора хранить массивом. Номера компонент отсчитываются от нуля

Конструктор вектора должен принимать число n – размерность вектора. Если n <= 0, то нужно бросать исключение IllegalArgumentException

Для этого класса нужно реализовать:

1. Конструкторы
   1. Vector(n) – размерность n, все компоненты равны 0
   2. Vector(Vector) – конструктор копирования
   3. Vector(double[]) – заполнение вектора значениями из массива
   4. Vector(n, double[]) – заполнение вектора значениями из массива. Если длина массива меньше n, то считать что в остальных компонентах 0
2. Метод getSize() для получения размерности вектора
3. Реализовать метод toString(), чтобы выдавал информацию о векторе в формате { значения компонент через запятую }  
   Например, { 1, 2, 3 }
4. Реализовать нестатические методы:
   1. Прибавление к вектору другого вектора
   2. Вычитание из вектора другого вектора
   3. Умножение вектора на скаляр
   4. Разворот вектора (умножение всех компонент на -1)
   5. Получение длины вектора
   6. Получение и установка компоненты вектора по индексу
   7. Переопределить метод equals, чтобы был true ⬄ векторы имеют одинаковую размерность и соответствующие компоненты равны. Соответственно, переопределить hashCode
5. Реализовать статические методы – должны создаваться новые векторы:
   1. Сложение двух векторов
   2. Вычитание векторов
   3. Скалярное произведение векторов

Если операции выполняются над векторами разной размерности, то считать что у меньшего вектора в недостающих компонентах нули

В main написать небольшую тестовую программу

Чему научитесь:

* Реализация достаточно сложного класса с многими операциями
* Тестирование за собой
* Поймете, что методы зачастую независимы друг от друга, и их можно писать и проверять по-отдельности. Но в некоторых случаях методы должны быть согласованы между собой
* Начальные навыки работы с исключениями
* Часто надежнее делать копию данных, чтобы не было неожиданностей со ссылочными типами

**Задача 3.** Матрица

Реализовать класс матрицы Matrix с использованием класса Vector – хранить строки как массив векторов

Реализовать:

1. Конструкторы:
   1. Matrix(n, m) – матрица нулей размера nxm
   2. Matrix(Matrix) – конструктор копирования
   3. Matrix(double[][]) – из двумерного массива
   4. Matrix(Vector[]) – из массива векторов-строк
2. Методы:
   1. Получение размеров матрицы
   2. Получение и задание вектора-строки по индексу
   3. Получение вектора-столбца по индексу
   4. Транспонирование матрицы
   5. Умножение на скаляр
   6. Вычисление определителя матрицы
   7. toString определить так, чтобы результат получался в таком виде: { { 1, 2 }, { 2, 3 } }
   8. умножение матрицы на вектор
   9. Сложение матриц
   10. Вычитание матриц
3. Статические методы:
   1. Сложение матриц
   2. Вычитание матриц
   3. Умножение матриц

Чему научитесь:

* Создание класса на основе другого своего класса
* Некоторые операции здесь сложны алгоритмически

**Задача 4.** List

1. Сделать классы для односвязного списка и узла списка.

Для эффективности сделайте поле для хранения длины списка.

Надо реализовать методы:

* получение размера списка
* получение значение первого элемента
* получение/изменение значения по указанному индексу.   
  Изменение значения по индексу пусть выдает старое значение.
* удаление элемента по индексу, пусть выдает значение элемента
* вставка элемента в начало
* вставка элемента по индексу
* удаление узла по значению, пусть выдает true, если элемент был удален
* удаление первого элемента, пусть выдает значение элемента
* разворот списка за линейное время
* копирование списка

2\* (Эта задача не обязательная). Есть односвязный список, каждый элемент которого хранит дополнительную ссылку на произвольный элемент списка. Эта ссылка может быть и null.

Надо создать копию этого списка, чтобы в копии эти произвольные ссылки ссылались на соответствующие элементы в копии.

Чему научитесь:

* Понимание односвязных списков
* Это достаточно сложно алгоритмически
* Generic’и

**Задача 5.** ArrayList

Сделать свою реализацию списка ArrayList<T> (в C# - List<T>), сделать его generic’ом.

Необходимо реализовать интерфейс List<T> (в C# - IList<T>).

Нужно реализовать специфичный конструктор, принимающий вместимость, а также методы ensureCapacity и trimToSize.

В Java методы sublist и listIterator реализовывать не нужно.

Чему научитесь:

* Понимание работы списка на массиве
* Реализация сложного интерфейса
* Реализация итератора
* Больше будете думать о временных оценках
* Generic’и

**Задача 6**. HashTable

Сделать свою реализацию хэш-таблицы, сделать ее generic’ом.

Класс должен реализовать интерфейс Collection<T>.

И в одном из конструкторов сделать параметр, который задает размер массива хэш-таблицы.

Чему научитесь:

* Понимание работы хэш-таблицы
* Реализация интересного итератора
* Generic’и

**Задача 7.** Сделать программу на Swing / Windows Forms для перевода температуры из одной шкалы в другую.

Необходимая функциональность:

1. Ввод температуры в поле ввода
2. Должна быть кнопка, которая переводит температуру из одной шкалы в другую
3. Результат перевода должен выводиться на форму, при этом быть не редактируемым
4. Можно задать из какой шкалы и в какую переводить
5. Доступные шкалы: цельсия, фаренгейта, кельвина
6. Если ввели не число, то нужно вывести ошибку
7. Обязательно использовать layout manager’ы

Научитесь:

* Начальное знакомство с UI
* MVC и отделение логики от представления
* Принцип открытости-закрытости

**Задача 8.** Сапер  
Условие задачи в отдельном файле

Чему научитесь:

* Писать сложное UI приложение с большим числом классов
* Анализировать существующий продукт (сапер в Windows)
* Находить нужную информацию в интернете
* Некоторые части сложны алгоритмически

**Задача 9 \*.** Tree

Реализовать бинарное дерево поиска.

Нужны операции:

1. Вставка
2. Поиск узла
3. Удаление первого вхождения узла по значению
4. Получение числа элементов
5. Обходы в ширину и глубину. Обход в глубину двумя вариантами – с рекурсией и без

Чему научитесь:

* Понимание работы с деревьями
* Обходы в глубину и ширину

**Задача 10 \*.** Graph

Реализовать обход несвязного графа в ширину и глубину.

Граф просто задаем двумерным массивом.

Чтобы реализовать обходы для несвязного графа надо просто посмотреть массив visited после обхода очередной компоненты связности.

Если остались не visited вершины, запускаем алгоритм для них и т.д., пока такие вершины не кончатся.

Чему научитесь:

* Понимание как работать с графами